|Scaler_Clustering_Analysis.ipynb| https://colab.research.google.com/drive/1Q_gLCFXvKudBEe1-g-omu8ufk0XQrYya?usp=sharing ||[github link]|https://github.com/rano667/Scaler-business-case-profiling-the-best-companies-and-job-positions|

1. EDA

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1.1 Load and inspect

```
df = pd.read_csv('/content/scaler_clustering.csv', index_col=0)

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 205843 entries, 0 to 206922
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   company_hash      205799 non-null  object
 1   email_hash        205843 non-null  object
 2   orgyear           205757 non-null  float64
 3   ctc               205843 non-null  int64
 4   job_position      153279 non-null  object
 5   ctc_updated_year  205843 non-null  float64
dtypes: float64(2), int64(1), object(3)
memory usage: 11.0+ MB
```

1.2 Missing values

```
print("\nMissing values per column:\n", df.isna().sum())


Missing values per column:
 company_hash          44
email_hash             0
orgyear               86
ctc                    0
job_position       52564
```

```
ctc_updated_year       0
dtype: int64
```

Step 1.1–1.2: Basic Data Overview

Dataset Shape & Structure

- **Shape**: 205,843 rows × 6 columns
- **Data Types**:
  - `company_hash`, `email_hash`, `job_position` → *object*
  - `orgyear`, `ctc_updated_year` → *float64*
  - `ctc` → *int64*

Missing Values

Column	Nulls	% Missing
`company_hash`	44	0.02%
`email_hash`	0	0.00%
`orgyear`	86	0.04%
`ctc`	0	0.00%
`job_position`	**52,564**	**25.5%**
`ctc_updated_year`	0	0.00%

> **Observation**: The major concern is the `job_position` column, with ~25% missing values. We'll need a strong imputation or exclusion strategy for this variable in the next preprocessing step.

Handle Missing job_position Values

```
# df['job_position_imputed'] = df['job_position'].isna().astype(int)

# Step 1: Fill missing job_position from other records of same email_hash
email_mode_job = df.groupby('email_hash')['job_position'].agg(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)
df['job_position'] = df.apply(
    lambda row: email_mode_job[row['email_hash']] if pd.isna(row['job_position']) else row['job_position'],
    axis=1
)

# Step 2: Fill remaining nulls using company mode
company_mode_job = df.groupby('company_hash')['job_position'].agg(lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan)
df['job_position'] = df.apply(
    lambda row: company_mode_job[row['company_hash']] if
```

```
pd.isna(row['job_position']) and not pd.isna(row['company_hash']) else
row['job_position'],
    axis=1
)

# Step 3: Fallback to 'Other'
df['job_position'] = df['job_position'].fillna('Other')
```

Apply KNN Imputer separately within each job group

(so comparisons happen among same roles)

```
from sklearn.impute import KNNImputer

def knn_impute_within_group(df, group_col, target_cols,
n_neighbors=5):
    df_result = df.copy()
    for grp, subset in df.groupby(group_col):
        imputer = KNNImputer(n_neighbors=n_neighbors)
        sub_imputed = imputer.fit_transform(subset[target_cols])
        df_result.loc[subset.index, target_cols] = sub_imputed
    return df_result

# Columns to impute
target_cols = ['orgyear', 'ctc', 'ctc_updated_year']

# Apply KNN within job_position group
df_imputed = knn_impute_within_group(df, 'job_position', target_cols)
df[['orgyear', 'ctc', 'ctc_updated_year']] = df_imputed[target_cols]

# rounding of
df['orgyear'] = df['orgyear'].round(0).astype(int)
```

Impute company_hash

 Why This Works:

- A learner (email_hash) usually stays in one company at a time (or has limited transitions).
- So the most frequent company_hash per email_hash is a reasonable proxy.

```
# Optional: Track Imputation
# df['company_imputed'] = df['company_hash'].isna().astype(int)

# Step 1: Most common company per learner
email_mode_company = df.groupby('email_hash')
['company_hash'].agg(lambda x: x.mode().iloc[0] if not x.mode().empty
else np.nan)
```

```python
# Step 2: Fill missing company_hash from other records of same
email_hash
df['company_hash'] = df.apply(
    lambda row: email_mode_company[row['email_hash']] if
pd.isna(row['company_hash']) else row['company_hash'],
    axis=1
)

# Step 3: Fallback to 'Unknown'
df['company_hash'] = df['company_hash'].fillna('Unknown')

df.isna().sum()
```

```
company_hash        0
email_hash          0
orgyear             0
ctc                 0
job_position        0
ctc_updated_year    0
dtype: int64
```

Data Preprocessing Complete

We've successfully imputed all missing values:

Feature	Imputation Strategy
`job_position`	From same `email_hash` → `company_hash` → 'Other'
`company_hash`	From same `email_hash` → 'Unknown'
`orgyear`	KNN Imputer within each `job_position` group
`ctc` + `ctc_updated_year`	KNN Imputed jointly with `orgyear`

> All missing values are now handled. We're ready to move to **CTC trajectory analysis** and **manual clustering**.

```python
df.sample(10)[['company_hash', 'job_position']]
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"company_hash\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"xzzgnxwvr ogrhnxgzo ucn rna\",\n
\"ojbvzntw\",\n          \"rtsvng ytvrny ntwyzgrgsxtovznytb xzw\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"job_position\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"Backend
```

```
Architect\",\n          \"Android Engineer\",\n          \"Other\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}

# # Clean strings using regex
# import re
# def clean_text(s):
#     if pd.isna(s): return s
#     return re.sub(r'[^A-Za-z0-9 ]+', '', s).strip()

# df['company_clean'] = df['company_hash'].apply(clean_text)
# df['job_position_clean'] = df['job_position'].apply(clean_text)
```

Regex Cleanup — Evaluation

After sampling multiple rows of `company_hash` and `job_position`, I observed that:

- `company_hash` values are anonymized hash strings (lowercase a–z only), no special characters
- `job_position` values are clean, readable job titles (e.g., "FullStack Engineer", "Backend Engineer")

Conclusion:

Regex cleanup is **not required** for either column, and skipping it preserves the integrity of already clean text data.

1.3 Unique Email_hash frequency

Why Check `Email_hash` Frequency?

`Email_hash` is a **proxy for the learner (person)**. Since it's anonymized PII, it helps us identify **unique individuals** in the dataset.

This step helps us:

1. **Detect Duplicate Records for the Same Person**

If one `Email_hash` appears multiple times:

- That person might have changed jobs
- Or been promoted (CTC updated)
- Or was mistakenly entered more than once

Insight: Multiple rows for the same `Email_hash` can indicate a **career trajectory**, not necessarily a data issue — but we should be aware of it for accurate clustering.

---

2. **Data Integrity Check**

If some `Email_hash` entries appear **too frequently (e.g. >10 times)**, it may be:

- A data entry problem (e.g. a system bug duplicating records)
- An edge case to handle separately (like contractors or reskilled alumni with multiple stints)

---

3. **Influences Manual Clustering**

If we're computing flags (like `Tier`, `Class`, or `Designation`) and the same person has multiple job entries:

- We might want to **use only the latest one**
- Or aggregate their records to see **progression trends**

```
email_freq = df['email_hash'].value_counts().reset_index()
email_freq.columns = ['email_hash','count']
print("\nTop duplicate email_hashes:\n",
email_freq[email_freq['count']>1].head())


Top duplicate email_hashes:
                                           email_hash  count
0  bbace3cc586400bbc65765bc6a16b77d8913836cfc98b7...     10
1  3e5e49daa5527a6d5a33599b238bf9bf31e85b9efa9a94...      9
2  298528ce3160cc761e4dc37a07337ee2e0589df251d736...      9
3  6842660273f70e9aa239026ba33bfe82275d6ab0d20124...      9
4  d598d6f1fb21b45593c2afc1c2f76ae9f4cb7167156cdf...      8
```

Duplicate `email_hash` Summary

Upon checking the frequency of `email_hash` (which represents anonymized learners), we observed that several individuals appear multiple times in the dataset:

Rank	`email_hash` (truncated)	Count
1	bbace3cc586400...	10
2	3e5e49daa5527a...	9
3	298528ce3160cc...	9
4	6842660273f70e...	9
5	d598d6f1fb21b4...	8

🔍 What This Means:

- These records likely reflect **career progression**, **CTC updates**, or **job changes** for the same learner over time.
- We **should not drop** these duplicates blindly.
- Instead, we can:
  - Use **latest record only** (based on `ctc_updated_year`) for clustering
  - Or retain all records if we want to **study trajectories** (e.g. job movement, promotions)
- These learners might be **highly engaged or successful**, worth flagging during profiling.

Next Step Recommendation:

We will extract and explore these learners' full history (`email_hash` + `ctc` + `job_position` + `ctc_updated_year`) to analyze growth or identify standout performers.

Analyze CTC progression over time for each learner

```
# get the count of unique email_hash in the data.

num_unique_emails = df['email_hash'].nunique()
print(f"\nNumber of unique email_hashes: {num_unique_emails}")
```

```
Number of unique email_hashes: 153443
```

```
#get unique company  hash

unique_companies = df['company_hash'].nunique()
print(f"\nNumber of unique company_hashes: {unique_companies}")
```

```
Number of unique company_hashes: 37300
```

```
# Step 1: Sort rows by learner and time
df_sorted = df.sort_values(by=['email_hash', 'ctc_updated_year'])

# Step 2: Calculate CTC progression and change flags
df_sorted['prev_ctc'] = df_sorted.groupby('email_hash')
['ctc'].shift(1)
df_sorted['ctc_growth_rate'] = ((df_sorted['ctc'] -
df_sorted['prev_ctc']) / df_sorted['prev_ctc']).round(2)

# Step 3: Track job & company changes
df_sorted['prev_company'] = df_sorted.groupby('email_hash')
['company_hash'].shift(1)
df_sorted['company_switch'] = (df_sorted['company_hash'] !=
df_sorted['prev_company']).astype(int)

df_sorted['prev_position'] = df_sorted.groupby('email_hash')
```

```
['job_position'].shift(1)
df_sorted['position_switch'] = (df_sorted['job_position'] !=
df_sorted['prev_position']).astype(int)

# Step 4: Binary flags for major CTC jumps (e.g. > 50%)
df_sorted['big_jump'] = (df_sorted['ctc_growth_rate'] >
0.5).astype(int)
```

CTC Trajectory Analysis

We analyzed how CTC changes over time for each learner using the following steps:

- Grouped records by `email_hash`
- Sorted by `ctc_updated_year`
- Calculated:
  - `ctc_growth_rate` — percentage change in salary
  - `company_switch` — whether the learner changed companies
  - `position_switch` — whether the learner changed job roles
  - `big_jump` — flagged CTC jumps > 50%

These trajectory signals will help us rank learners and assign clustering flags based on performance and mobility.

Add a trajectory classification label:

```
def classify_growth(rate):
    if pd.isna(rate):
        return 'unknown'
    elif rate > 0.5:
        return 'aggressive'
    elif rate > 0.1:
        return 'steady'
    else:
        return 'flat'

df_sorted['growth_type'] =
df_sorted['ctc_growth_rate'].apply(classify_growth)
```

Count overall switches

```
switch_summary = df_sorted.groupby('email_hash').agg({
    'company_switch': 'sum',
    'position_switch': 'sum',
    'big_jump': 'sum'
}).reset_index()
```

Build Clustering Flags

1. Designation Flag

Group by: company_hash, job_position, orgyear Compare: Individual CTC vs avg CTC in same company-role-year group

```
# Group-level stats
designation_stats = df.groupby(['company_hash', 'job_position',
'orgyear'])['ctc'].agg(['mean', 'std']).reset_index()
designation_stats.rename(columns={'mean': 'avg_ctc'}, inplace=True)

# Merge back
df = df.merge(designation_stats, on=['company_hash', 'job_position',
'orgyear'], how='left')

# Create flag
df['designation'] = df.apply(
    lambda row: 1 if row['ctc'] > row['avg_ctc'] else (3 if row['ctc']
< row['avg_ctc'] else 2),
    axis=1
)
```

2. Class Flag

Group by: company_hash, job_position Compare: CTC vs avg CTC in same company-role

```
class_stats = df.groupby(['company_hash', 'job_position'])
['ctc'].mean().reset_index().rename(columns={'ctc': 'class_avg_ctc'})
df = df.merge(class_stats, on=['company_hash', 'job_position'],
how='left')

df['class'] = df.apply(
    lambda row: 1 if row['ctc'] > row['class_avg_ctc'] else (3 if
row['ctc'] < row['class_avg_ctc'] else 2),
    axis=1
)
```

1. Tier Flag

Group by: company_hash Compare: CTC vs avg CTC in the company

```
tier_stats = df.groupby('company_hash')
['ctc'].mean().reset_index().rename(columns={'ctc': 'tier_avg_ctc'})
df = df.merge(tier_stats, on='company_hash', how='left')

df['tier'] = df.apply(
    lambda row: 1 if row['ctc'] > row['tier_avg_ctc'] else (3 if
row['ctc'] < row['tier_avg_ctc'] else 2),
```

```
    axis=1
)

df.drop(columns=['avg_ctc', 'class_avg_ctc', 'tier_avg_ctc'],
inplace=True)

df.head()
```

{"type":"dataframe","variable_name":"df"}

□ Manual Clustering: Designation, Class, and Tier Flags

We manually grouped learners into performance buckets by comparing their CTC with their peers:

Flag Name	Grouping Level	Purpose
`designation`	`company_hash`, `job_position`, `orgyear`	Compares salary within same job & year
`class`	`company_hash`, `job_position`	Compares salary within same job role
`tier`	`company_hash`	Compares salary across the company

Each flag uses:

- `1` → Above average
- `2` → Around average
- `3` → Below average

□ 1. Top 10 Employees Earning the Most in Tier 1 Companies

```
top_tier1_employees = df[df['tier'] == 1].sort_values(by='ctc',
ascending=False).head(10)
```

□ 2. Top 10 Data Science Employees in Each Company (Class 1)

```
# Filter for DS roles — adjust keywords as needed
ds_roles = ['data', 'ml', 'ai']
df['is_data_role'] =
df['job_position'].str.lower().str.contains('|'.join(ds_roles))

top_ds_by_company = (
    df[(df['class'] == 1) & (df['is_data_role'])]
    .sort_values(by='ctc', ascending=False)
    .groupby('company_hash')
    .head(10)
)
```

3. Bottom 10 Data Science Employees (Class 3)

```
bottom_ds_by_company = (
    df[(df['class'] == 3) & (df['is_data_role'])]
    .sort_values(by='ctc', ascending=True)
    .groupby('company_hash')
    .head(10)
)
```

4. Bottom 10 Employees in Tier 3 Companies

```
bottom_tier3_employees = df[df['tier'] == 3].sort_values(by='ctc',
ascending=True).head(10)
```

5. Top 10 Employees in Each Company with 5/6/7 YOE (High Earning in Their Tier)

```
target_years = [2018, 2019, 2020]
top_mid_exp_by_company = (
    df[df['orgyear'].isin(target_years)]
    .sort_values(by='ctc', ascending=False)
    .groupby('company_hash')
    .head(10)
)
```

6. Top 10 Companies by Average CTC

```
top_companies_by_ctc = df.groupby('company_hash')
['ctc'].mean().reset_index().sort_values(by='ctc',
ascending=False).head(10)
```

7. Top 2 Job Positions in Every Company by Average CTC

```
top_roles_by_company = (
    df.groupby(['company_hash', 'job_position'])['ctc'].mean()
    .reset_index()
    .sort_values(['company_hash', 'ctc'], ascending=[True, False])
    .groupby('company_hash')
    .head(2)
)

results = {
    'top_tier1_employees': top_tier1_employees,
    'top_ds_by_company': top_ds_by_company,
    'bottom_ds_by_company': bottom_ds_by_company,
    'bottom_tier3_employees': bottom_tier3_employees,
    'top_mid_exp_by_company': top_mid_exp_by_company,
    'top_companies_by_ctc': top_companies_by_ctc,
    'top_roles_by_company': top_roles_by_company,
}
```

```python
print(results)
```

```
{'top_tier1_employees':                              company_hash  \
117636               obvqnuqxdwgb
82611        nvnv ntrtotqcxwto rna
82601                   xwxwx mvzp
165596               tdutaxv sqghu
3471                   zgn fgqpxzs
82539                   eqttwyvqst
82499   ytfrtnn uvwpvqa tzntquqxot
12749                ntvb wgbuhntqo
82584                    yaew mvzp
86738                zgn vuurxwvmrt

                                              email_hash  orgyear
ctc  \
117636  5b4bed51797140db4ed52018a979db1e34cee49e27b488...     2018
255555555
82611   e3ef9223ad1dd7385e7344270c1b1ecee22ab22da0d52c...     2010
200000000
82601   2311bf023218afe93d650cac03abb7a40f7fa55c08d260...     2018
200000000
165596  55ce75df4e43b5ee10d59f34b2dffd5c1ee6170f2d38c4...     2017
200000000
3471    6d3f1c57e8840cd379c472b4cf4847c1330f12ae76a55e...     2019
200000000
82539   4c19cfc1aa47a5b007004fadeacb88da76b6a59ff4271f...     1998
200000000
82499   f195ae4e02da9f187009f8545061a65f8a22a99c0e7aeb...     2018
200000000
12749   6739ade7083af0779d5eb4cf40bcb9dce42d314fb234c2...     2015
200000000
82584   1ad3d8c2b855e9cbe6bb187e8140a07ed8a8b29d275ae0...     2017
200000000
86738   d7419e4ad5de93adc08aef91889ead458b247ec41bb889...     2014
200000000

               job_position  ctc_updated_year          std
designation  \
117636   FullStack Engineer            2016.0  1.268291e+08
1
82611                 Other            2020.0           NaN
2
82601                 Other            2020.0  6.651215e+07
1
165596     Backend Engineer            2019.0  5.102590e+07
1
3471                  Other            2020.0  7.039280e+07
1
82539   Security Leadership            2020.0           NaN
```

```
2
82499                 Other            2020.0  1.145645e+08
1
12749    FullStack Engineer            2020.0           NaN
2
82584                 Other            2020.0  7.051566e+07
1
86738      Backend Engineer            2021.0  4.768151e+07
1

        class  tier
117636      1     1
82611       1     1
82601       1     1
165596      1     1
3471        1     1
82539       2     1
82499       1     1
12749       1     1
82584       1     1
86738       1     1  , 'top_ds_by_company':
company_hash  \
836     mqxonrtwgzt v bvyxzaqv sqghu wgbuvzj
19712              nvnv wgzohrnvzwj otqcxwto
909                               wgszxkvzn
18899              qtrxvzwt xzegwgbb rxbxnta
33984                           ogwxn szqvrt
...                                      ...
48811                                 mvjtq
183682               zgqnytvontqz hzxctqoxnj
200349               zgqnytvontqz hzxctqoxnj
202441               zgqnytvontqz hzxctqoxnj
153496            exznqhon ogrhnxgzo ucn rna

                                               email_hash  orgyear
ctc  \
836     cda8d723438e81185d2ee8c348870a4612eea974cdb2db...     2017
200000000
19712   59316048d113539202325e05af9b66620255ba84eab635...     2015
200000000
909     aad581a532f319c76c6e73937572feed9867d5ee2f1093...     2014
200000000
18899   f1b31a501f6b7fd6edae9e7e883bf60d2d3bff0fa37368...     2017
200000000
33984   f5b2a30853a67e1703249db6003884d7e1ae69e0c03aa0...     2014
200000000
...                                                   ...      ...
...
48811   db2c70fea469a7f1456457812fe94a01c337eb6ce75bd5...     2018
```

```
115000
183682  47073bf02fd4a41a6b446e8422e5ce427fc824eb3d6700...     2012
52000
200349  66570945ebc4bb7dad62ceeff1ff0fff3c007e79146143...     2021
47000
202441  66570945ebc4bb7dad62ceeff1ff0fff3c007e79146143...     2021
47000
153496  ab2dc9db23c3104f0b6b3dbd4cdd5bfb9e5829b8b7943d...     2017
10000

          job_position  ctc_updated_year           std  designation
class  \
836     Data Scientist            2020.0           NaN            2
1
19712     Data Analyst            2020.0  6.741396e+07            1
1
909       Data Analyst            2020.0  4.374237e+07            1
1
18899     Data Analyst            2020.0           NaN            2
1
33984     Data Analyst            2020.0  0.000000e+00            2
1
...                ...               ...           ...          ...
...
48811   Data Scientist            2019.0           NaN            2
1
183682    Data Analyst            2015.0           NaN            2
1
200349    Data Analyst            2020.0  0.000000e+00            2
1
202441    Data Analyst            2020.0  0.000000e+00            2
1
153496  Data Scientist            2020.0  1.979899e+03            1
1

        tier  is_data_role
836        1          True
19712      1          True
909        1          True
18899      1          True
33984      1          True
...      ...           ...
48811      3          True
183682     1          True
200349     1          True
202441     1          True
153496     1          True

[2086 rows x 11 columns], 'bottom_ds_by_company':
company_hash  \
```

```
147805              zgn vuurxwvmrt vwwghzn
177550           nvnv wgzohrnvzwj otqcxwto
190300                     fxootz xzegntwy
10835   srgmvrtast  xzntrrxstzwt ge nyxzso
8705                   bxyhu wgbbhzxwvnxgz
...                                    ...
54283                               xznhxn
92858                           fgqpehoxgz
89151                            yaew rxet
17448                            xzattawgb
107038                              wvqttb

                                               email_hash  orgyear
ctc  \
147805  299f764fcae62f331f3c5eb1b451e7107302ded46e2a71...     2007
1000
177550  3becd3658bc0d426f8867142eb6cbd7e9ca9f43b572794...     2018
3500
190300  9810176ff1b7312a460834736ac273104d5152d3ded540...     2005
3800
10835   8001bc017fbe95541d23f5780c3edb988b7d9b2225e39e...     2017
4000
8705    690f6fdab1ab7514a6a9325ebd6cfe910dbf12d46b6fde...     2018
4000
...                                                   ...      ...
...
54283   7adf15ce2bfe62c24f197f0a0499b47ed93b433300db35...     2010
3500000
92858   6f7c8da2e0d377d85a59c64724c73930277065c1510262...     2011
3900000
89151   3d3685ed8b43efc9e478920c55ad9e62b8c7aded2261cd...     1997
4000000
17448   0442787ae22a16022131f18e10e689aca9bfbb193713a7...     2016
4800000
107038  0485990d28fdbb10e494793b31dd97f94c326a93c07a2d...     2014
7000000

                  job_position  ctc_updated_year           std
designation  \
147805            Data Analyst            2021.0           NaN
2
177550  Database Administrator            2019.0  5.762313e+07
3
190300  Database Administrator            2019.0           NaN
2
10835           Data Scientist            2019.0           NaN
2
8705            Data Scientist            2019.0           NaN
2
```

```
...                          ...                ...           ...
...
54283           Data Scientist            2018.0  1.767767e+06
1
92858           Data Scientist            2016.0           NaN
2
89151           Data Scientist            2019.0           NaN
2
17448           Data Scientist            2019.0  5.656854e+05
3
107038          Data Scientist            2020.0           NaN
2

        class  tier  is_data_role
147805      3     3          True
177550      3     3          True
190300      3     3          True
10835       3     3          True
8705        3     3          True
...       ...   ...           ...
54283       3     1          True
92858       3     1          True
89151       3     3          True
17448       3     1          True
107038      3     3          True

[2453 rows x 11 columns], 'bottom_tier3_employees':
company_hash  \
135435               xzntqcxtfmxn
118236               xzntqcxtfmxn
114164               xzntqcxtfmxn
184946                         xm
116946   hzxctqoxnj ge fvoyxzsngz
150682                        zvz
99419                         gjg
171196  nvnv wgzohrnvzwj otqcxwto
159689               kvrgqv sqghu
179326            mtznrtj ojontbo

                                               email_hash  orgyear
ctc  \
135435  3505b02549ebe2c95840ac6f0a35561a3b4cbe4b79cdb1...     2014
2
118236  f2b58aeed3c074652de2cfd3c0717a5d21d6fbcf342a78...     2013
6
114164  23ad96d6b6f1ecf554a52f6e9b61677c7d73d8a409a143...     2013
14
184946  b8a0bb340583936b5a7923947e9aec21add5ebc50cd60b...     2016
15
116946  f7e5e788676100d7c4146740ada9e2f8974defc01f571d...     2022
```

```
200
150682  9af3dca6c9d705d8d42585ccfce2627f00e1629130d14e...     2023
600
99419   b995d7a2ae5c6f8497762ce04dc5c04ad6ec734d70802a...     2018
600
171196  80ba0259f9f59034c4927cf3bd38dc9ce2eb60ff18135b...     2012
600
159689  ae625c7063c1f8194deadfb28905d5dcc6f9077274a083...     2017
1000
179326  7c8e0d8194db4deb41cbc9b3b6c428e0f9ab289436638e...     2016
1000

              job_position  ctc_updated_year           std
designation  \
135435    Backend Engineer            2019.0  1.154699e+06
3
118236    Backend Engineer            2018.0  3.868189e+05
3
114164    Backend Engineer            2018.0  3.868189e+05
3
184946               Other            2018.0           NaN
2
116946    Backend Engineer            2021.0           NaN
2
150682    Backend Engineer            2019.0  2.067703e+06
3
99419   FullStack Engineer            2021.0  6.078993e+05
3
171196    Backend Engineer            2017.0  1.215077e+07
3
159689    Backend Engineer            2021.0  2.757716e+04
3
179326  FullStack Engineer            2019.0           NaN
2

        class  tier  is_data_role
135435      3     3         False
118236      3     3         False
114164      3     3         False
184946      3     3         False
116946      2     3         False
150682      3     3         False
99419       3     3         False
171196      3     3         False
159689      3     3         False
179326      3     3         False  , 'top_mid_exp_by_company':
company_hash  \
117636                                    obvqnuqxdwgb
22973                                      vznxzg rvmo
```

```
23068                                           egdwgzz
82601                                        xwxwx mvzp
205614                                   zgn vuurxwvmrt
...                                                 ...
96913   hzxctqoxava wtznqvr bvqnv vmqth at rvo cxrrvo
151402                                           tzavcv
82028                                    uqgmrtb ogrcxzs
54820                                uqvpqxnx voogwxvnto
183804                                               xm

                                               email_hash  orgyear
ctc  \
117636  5b4bed51797140db4ed52018a979db1e34cee49e27b488...     2018
255555555
22973   634fd283565b8954513a6ad0e47cedb0fa8847923149fb...     2019
200000000
23068   05a0aec8db0d251460a2e200d835b7818e050308ef1e67...     2018
200000000
82601   2311bf023218afe93d650cac03abb7a40f7fa55c08d260...     2018
200000000
205614  48b00207f75dd25ca9d518103e2ddc3c9a9706e51ae393...     2020
200000000
...                                                   ...      ...
...
96913   a842673b5abebd7bf405bb7ad41560f6a2a586be2831c2...     2018
1000
151402  e451a317cbb3244e0571bec1d6a6e0919f53725b54a5f3...     2019
1000
82028   edcfb902656b736e1f35863298706d9d34ee795b7ed85a...     2018
500
54820   8786759b95d673466e94f62f1b15e4f8c6bd7de6164074...     2020
24
183804  75357254a31f133e2d3870057922feddeba82b88056a07...     2019
16

                  job_position  ctc_updated_year           std
designation  \
117636      FullStack Engineer            2016.0  1.268291e+08
1
22973        Frontend Engineer            2020.0           NaN
2
23068   Engineering Leadership            2020.0           NaN
2
82601                    Other            2020.0  6.651215e+07
1
205614       Frontend Engineer            2019.0  9.984172e+07
1
...                        ...               ...           ...
...
```

```
96913       FullStack Engineer           2019.0          NaN
2
151402            iOS Engineer           2019.0          NaN
2
82028               Co-founder           2019.0          NaN
2
54820                    Other           2020.0          NaN
2
183804                   Other           2018.0          NaN
2

        class  tier  is_data_role
117636      1     1         False
22973       1     1         False
23068       2     1         False
82601       1     1         False
205614      1     1         False
...       ...   ...           ...
96913       2     2         False
151402      2     2         False
82028       2     2         False
54820       2     2         False
183804      1     1         False

[26054 rows x 11 columns], 'top_companies_by_ctc':
company_hash           ctc
30495                    whmxw rgsxwo uqxcvnt rxbxnta  1.000150e+09
1219                aveegaxr xzntqzvnxgzvr hzxctqoxnj  2.500000e+08
19855                          qtygmgny tzsxzttqxzs  2.000000e+08
33218                         xtrrxuot ntwyzgrgsxto  2.000000e+08
21530                           sgraygbk wgzohrnxzs  2.000000e+08
19999               qvj mhoxztoo ntwyzgrgsxto ucn rna  2.000000e+08
15896                onvqrxnt btaxv vza tzntqnvxzbtzn  2.000000e+08
23546                     touxqxn ntwyzgrgsxto ucnrna  2.000000e+08
2394                      bjnqvy tztqsj xzaxv ucn rna  2.000000e+08
27511  vooxontvoj cxqnhvr onveexzs uqxcvnt ogrhnxgzo  2.000000e+08,
'top_roles_by_company':                          company_hash
job_position       ctc
0                                    0              Other   100000.0
1                                 0000              Other   300000.0
3                           01 ojztqsj   Frontend Engineer   830000.0
2                           01 ojztqsj    Android Engineer   270000.0
4      05mz exzytvrny uqxcvnt rxbxnta    Backend Engineer  1100000.0
...                                ...                 ...        ...
62471     zyvzwt wgzohrnxzs tzsxzttqo   Frontend Engineer   940000.0
62472                               zz              Other   935000.0
62473    zzb ztdnstz vacxogqj ucn rna  FullStack Engineer   600000.0
62474                           zzgato              Other   130000.0
62475                           zzzbzb              Other   720000.0
```

```
[45764 rows x 3 columns]}

df.shape

(205843, 11)
```

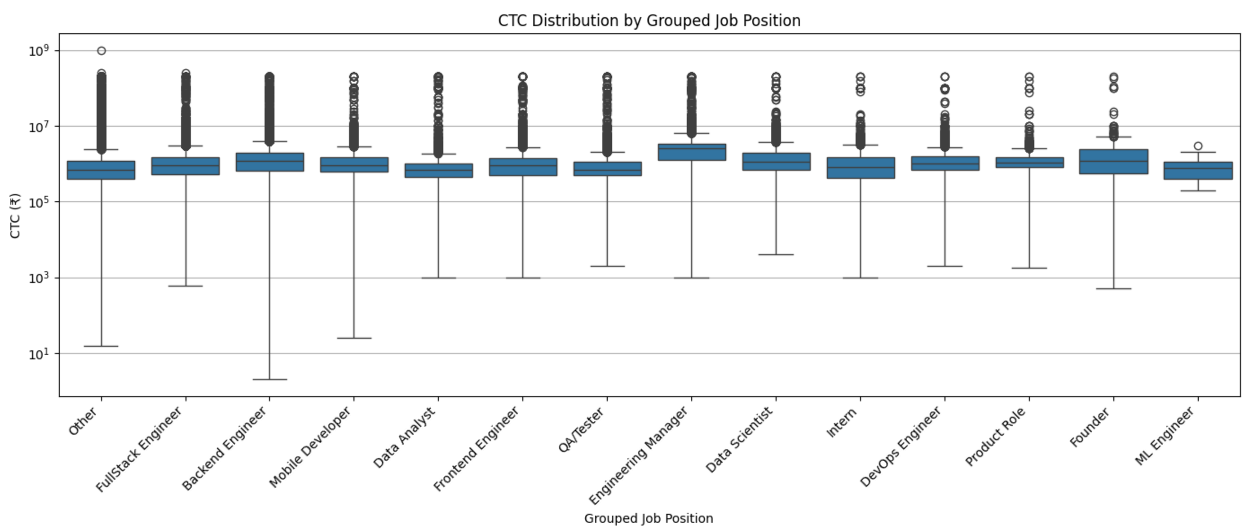
 1. Boxplot: CTC by Job Position

```python
# unique job positions
df['job_position'].nunique()

1016

import re

def normalize_job_title(title):
    if pd.isna(title):
        return 'Other'

    title_lower = title.lower()

    if 'fullstack' in title_lower:
        return 'FullStack Engineer'
    elif 'backend' in title_lower:
        return 'Backend Engineer'
    elif 'frontend' in title_lower or 'ui' in title_lower:
        return 'Frontend Engineer'
    elif 'data scientist' in title_lower:
        return 'Data Scientist'
    elif 'data analyst' in title_lower:
        return 'Data Analyst'
    elif 'ml engineer' in title_lower or 'machine learning' in
title_lower:
        return 'ML Engineer'
    elif 'devops' in title_lower or 'site reliability' in title_lower:
        return 'DevOps Engineer'
    elif 'mobile' in title_lower or 'android' in title_lower or 'ios'
in title_lower:
        return 'Mobile Developer'
    elif 'qa' in title_lower or 'test' in title_lower:
        return 'QA/Tester'
    elif 'lead' in title_lower or 'manager' in title_lower or 'head'
in title_lower:
        return 'Engineering Manager'
    elif 'product' in title_lower:
        return 'Product Role'
    elif 'intern' in title_lower:
        return 'Intern'
    elif 'founder' in title_lower or 'co-founder' in title_lower:
```

```python
        return 'Founder'
    else:
        return 'Other'

# Apply normalization
df['job_position_grouped'] =
df['job_position'].apply(normalize_job_title)

# unique job positions
df['job_position_grouped'].nunique()
```

14

```python
plt.figure(figsize=(14, 6))
sns.boxplot(data=df, x='job_position_grouped', y='ctc')
plt.xticks(rotation=45, ha='right')
plt.title('CTC Distribution by Grouped Job Position')
plt.ylabel('CTC (₹)')
plt.xlabel('Grouped Job Position')
plt.yscale('log')
plt.grid(True, axis='y')
plt.tight_layout()
plt.show()
```



Observations from CTC Distribution by Grouped Job Position

- **Engineering Manager**, **Founder**, and **DevOps Engineer** roles generally show **higher median CTCs**, indicating strong compensation for leadership or infrastructure-focused positions.
- **Intern**, **QA/Tester**, and **Data Analyst** roles have **lower median CTCs**, aligning with their entry-level or support nature.
- **FullStack**, **Backend**, and **Frontend Engineers** show **similar median CTC ranges**, with **Backend** slightly higher in some cases.

- The **"Other"** category has a **wide range and high variance**, suggesting it still contains **unclassified or noisy job titles**—potentially needing further cleaning or re-grouping.
- **ML Engineer** roles exhibit a **narrower distribution** but a **lower median**, possibly due to fewer data points or more junior-level entries in this group.

Custering

```
df.shape
```

```
(205843, 12)
```

￿ Step 1: Aggregate at Employee Level

```
# Step 1: Aggregating to unique employee level
employee_df = df.groupby('email_hash').agg({
    'company_hash': lambda x: x.mode().iloc[0] if not x.mode().empty
else 'Unknown',
    'job_position': lambda x: x.mode().iloc[0] if not x.mode().empty
else 'Other',
    'orgyear': 'min',
    'ctc': 'mean',
    'ctc_updated_year': 'max',
    'designation': 'max',
    'class': 'max',
    'tier': 'max'
}).reset_index()

# Step 2: Calculate Years of Experience
from datetime import datetime
current_year = datetime.now().year
employee_df['yoe'] = current_year - employee_df['orgyear']

# Check for negative yoe
negative_yoe_employees = employee_df[employee_df['yoe'] < 0]

if not negative_yoe_employees.empty:
    print("\nEmployees with negative Years of Experience:")
    print(negative_yoe_employees[['email_hash', 'orgyear', 'yoe']])
else:
    print("\nNo employees found with negative Years of Experience.")
```

```
Employees with negative Years of Experience:
                                              email_hash  orgyear
yoe
7927    0ceab34736c0ba43f541a9d62f5f8ffe33f4c306ea73a5...     2026
-1
26920   2cc6bae4e52677d27ce3fca38d7a01ecbe537e1dc1c48d...     2106
-81
```

```
30953   3394674bb6bb1de6289e931853fa0bd131c811e0054a92...     2031
-6
38406   4007e5caadc3f52c3e18bf2b4eacbadf17b114208c2d04...    20165 -
18140
49055   5221d938a36c77d13eb0c6c4242a3ae52c9a535951e18a...     2026
-1
63855   6aa38b497c73367a7dd6eafb95bdd5b07cca83ed14c588...     2026
-1
65962   6e2ce64fb85d30c8e82fd7a60fd1ca0768ab262b9ea31b...     2026
-1
67986   7191da2e57dcb0c1301711e889ea72d5cc801e039359b1...    20165 -
18140
69514   74348d9362f32b8ba7a8234b3d4cb29296e00dfbfbffa4...     2031
-6
72721   799dff77b331bfac04cf005935acf7e0d16845f4f24798...     2204   -
179
83471   8b82635f6d131631b1c1e1dad46d104d6e4573f9be77bd...     2029
-4
87381   91e4562ab8bab639b859082d519722a91b8e6f3d55c109...     2026
-1
88283   935cfbdb93bf206fbee0b6a0b0062abb387be8f249d1cf...     2029
-4
104972  af3f2d5aff40c73774a6f2a9f36502fbe298ebdc7834e4...     2026
-1
128476  d66f939c4318c1958be5bc9e7b70b741aa61be7493ff58...     2028
-3
128754  d6df76c2b61fa3a068e4e3812be12a58f86f78a31fe888...     2029
-4
129378  d7ebf59204eff38360edce9659282085ad2aaf51d5cdc3...     2029
-4
133591  df04fed73ba1e86a59f40279046bf0a1a5b8c0e98a85e1...     2027
-2
144306  f0c712df5b5e6698a7558311dff87d2b2b4aaa12839915...     2029
-4
147725  f648fa217922f5a36b510df6346a2041a3483e21289069...     2101
-76
152821  fee9df1faa9d4a38bb97185bb9af6687cba48b514f5d04...     2026
-1

from sklearn.impute import KNNImputer

# Step 1: Identify & Nullify invalid orgyear
# Assuming valid orgyear is between 1980 and current year (2025)
valid_orgyear_mask = (employee_df['orgyear'] >= 1980) &
(employee_df['orgyear'] <= 2025)
employee_df.loc[~valid_orgyear_mask, 'orgyear'] = np.nan

# Optional: Check how many were affected
print(f"Invalid orgyear replaced with NaN:
{employee_df['orgyear'].isna().sum()}")
```

```
# Step 2: Prepare data for KNN imputation
# Select only numeric columns needed for imputation (add others if
useful)
impute_cols = ['orgyear', 'ctc', 'ctc_updated_year']
impute_data = employee_df[impute_cols]

# KNN Imputer (k=5 is default)
imputer = KNNImputer(n_neighbors=5)
imputed_array = imputer.fit_transform(impute_data)

# Update dataframe with imputed values
employee_df[impute_cols] = pd.DataFrame(imputed_array,
columns=impute_cols)

# Round orgyear to nearest whole year after imputation
employee_df['orgyear'] = employee_df['orgyear'].round().astype(int)

# Step 3: Recalculate YOE (Years of Experience)
employee_df['yoe'] = 2025 - employee_df['orgyear']

# Step 4: (Optional) Final sanity check
# Set yoe to NaN if still invalid (e.g. negative or > 45 years of
experience)
employee_df.loc[(employee_df['yoe'] < 0) | (employee_df['yoe'] > 45),
'yoe'] = np.nan

# Final print to confirm cleanup
print(employee_df[['orgyear', 'yoe']].describe())
```

```
Invalid orgyear replaced with NaN: 71
             orgyear            yoe
count  153443.000000  153443.000000
mean     2014.808450      10.191550
std         4.356792       4.356792
min      1981.000000       0.000000
25%      2013.000000       7.000000
50%      2016.000000       9.000000
75%      2018.000000      12.000000
max      2025.000000      44.000000
```

```
# Check for negative yoe
negative_yoe_employees = employee_df[employee_df['yoe'] < 0]

if not negative_yoe_employees.empty:
    print("\nEmployees with negative Years of Experience:")
    print(negative_yoe_employees[['email_hash', 'orgyear', 'yoe']])
else:
    print("\nNo employees found with negative Years of Experience.")
```

```
No employees found with negative Years of Experience.
```

⚙ Step 2: Preprocessing for Clustering

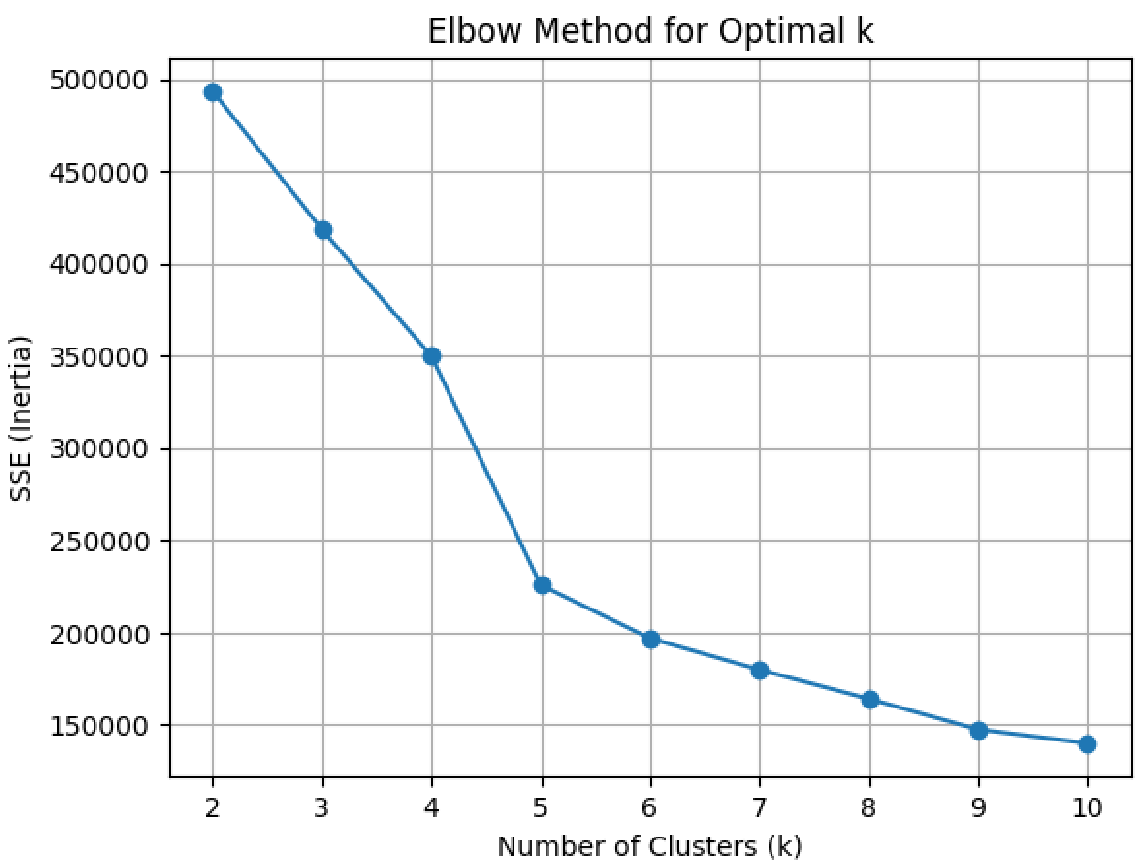
```
from sklearn.preprocessing import LabelEncoder, StandardScaler

# # Encode categorical variables
# label_cols = ['company_hash', 'job_position']
# for col in label_cols:
#     le = LabelEncoder()
#     employee_df[col] = le.fit_transform(employee_df[col])

# Step 1: Store separate encoders for each column
le_company = LabelEncoder()
le_job = LabelEncoder()

# Step 2: Fit and transform using those
employee_df['company_hash'] =
le_company.fit_transform(employee_df['company_hash'])
employee_df['job_position'] =
le_job.fit_transform(employee_df['job_position'])

# Step 3: Create reverse mapping from encoded value to actual value
job_position_mapping = dict(zip(range(len(le_job.classes_)),
le_job.classes_))

# Step 4: Add a column with actual job position names
employee_df['job_position_actual'] =
employee_df['job_position'].map(job_position_mapping)

# Select features for clustering
features = ['company_hash', 'job_position', 'ctc', 'yoe']

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(employee_df[features])
```

Step 3: Clustering Tendency Check

```
# !pip install pyclustertend
# from pyclustertend import hopkins
# import numpy as np

# # Hopkins statistic (closer to 0 = good tendency to cluster)
# hopkins_score = hopkins(X_scaled, len(X_scaled))
# print("Hopkins Score:", hopkins_score)

# # output: near zero -> 0.00001
```

□ Step 4: Elbow Method to Pick k

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

sse = []
k_range = range(2, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_scaled)
    sse.append(km.inertia_)

plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()
```



Step 5: Apply K-Means Clustering

```python
# Assuming optimal k is 5
kmeans = KMeans(n_clusters=5, random_state=42)
employee_df['cluster'] = kmeans.fit_predict(X_scaled)

employee_df.sample(10)
```

{"summary":"{\n \"name\": \"employee_df\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"email_hash\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"ecddd79012d5856159f0cb7e88c72382719d4253546dfae4df07f19c853413b9\",\n   \"83e62bc06240a1a86564f937ce703bedcbe0bc797f469d8fe4bb40406ea60aea\",\n   \"f90c0a6a0c8722adae5fa5c45cc93f6b030eb3b19101e701bed2f7b711d56a81\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"company_hash\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 11509,\n   \"min\": 2091,\n   \"max\": 35220,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   25039,\n   4893,\n   23583\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"job_position\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 202,\n   \"min\": 135,\n   \"max\": 706,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   256,\n   135,\n   628\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"orgyear\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4,\n   \"min\": 2004,\n   \"max\": 2018,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   2018,\n   2016,\n   2004\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1730626.4119612232,\n   \"min\": 320000.0,\n   \"max\": 6000000.0,\n   \"num_unique_values\": 9,\n   \"samples\": [\n   1500000.0,\n   3000000.0,\n   2900000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc_updated_year\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9944289260117533,\n   \"min\": 2019.0,\n   \"max\": 2021.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   2021.0,\n   2019.0,\n   2020.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"designation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n   \"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   2,\n   1,\n   3\n   ],\n

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"class\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\":
1,\n   \"max\": 3,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   3,\n   1,\n   2\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"tier\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   3,\n   1,\n   2\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"yoe\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 4.4671641514002545,\n
\"min\": 7.0,\n   \"max\": 21.0,\n   \"num_unique_values\":
7,\n   \"samples\": [\n   7.0,\n   9.0,\n
21.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"job_position_actual\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   \"FullStack Engineer\",\n   \"Backend Engineer\",\
n   \"Support Engineer\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"cluster\",\n   \"properties\":
{\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 4,\n
\"samples\": [\n   1,\n   3,\n   0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

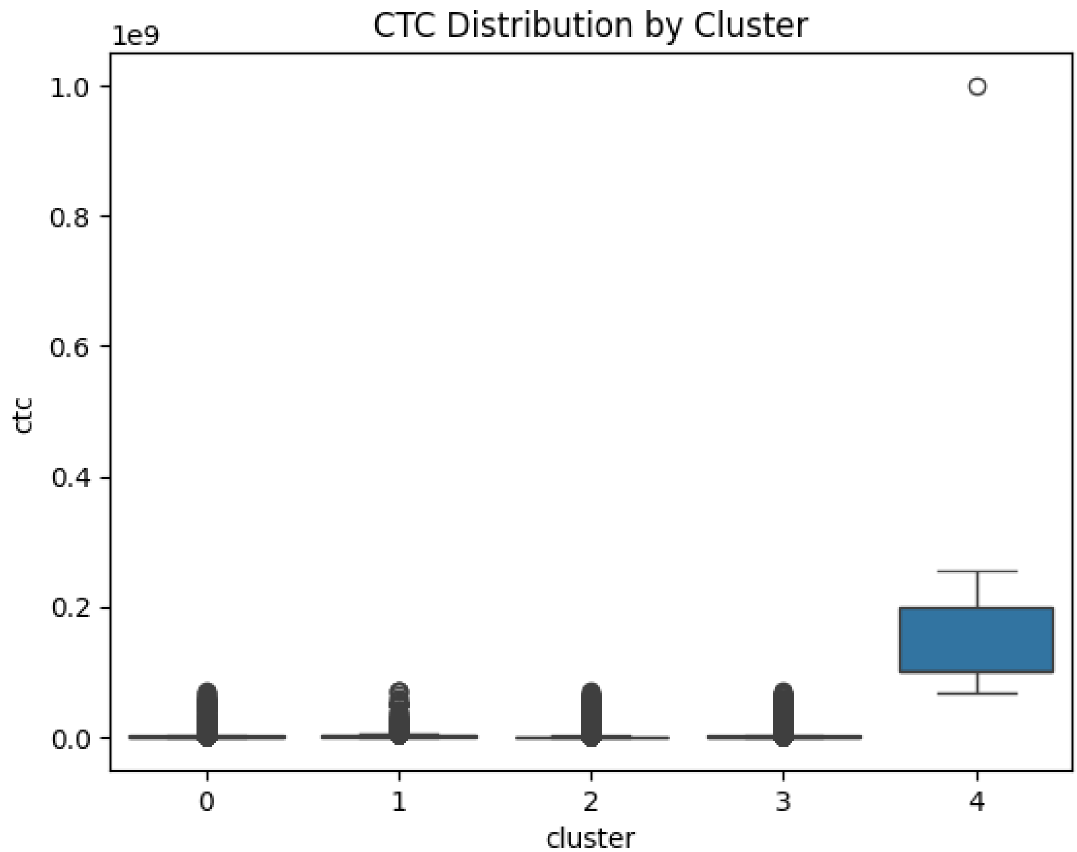
□ Step 1: Summary Statistics by Cluster

```
# Overview of each cluster
cluster_summary = employee_df.groupby('cluster').agg({
    'ctc': ['mean', 'median', 'min', 'max', 'count'],
    'yoe': ['mean', 'min', 'max'],
    'designation': 'mean',
    'class': 'mean',
    'tier': 'mean'
}).round(2)

cluster_summary.columns = ['_'.join(col) for col in
cluster_summary.columns]
cluster_summary.reset_index(inplace=True)
cluster_summary
```

{"summary":"{\n \"name\": \"cluster_summary\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"cluster\",\n
\"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   1,\n
4,\n   2\n   ],\n   \"semantic_type\": \"\",\n

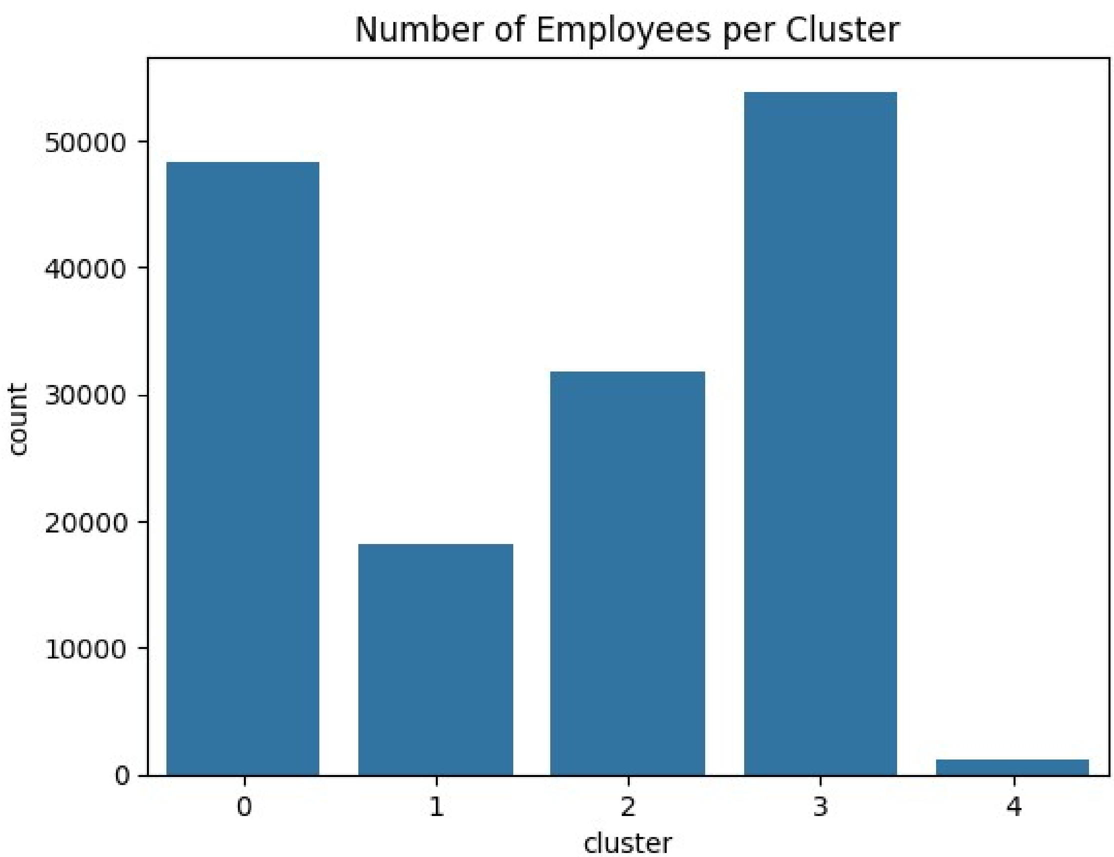
```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 59780678.47022072,\n        \"min\":
1156932.86,\n        \"max\": 135246034.29,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
2408983.83,\n          135246034.29,\n          1156932.86\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_median\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 44208813.82710918,\n        \"min\":
680000.0,\n        \"max\": 100000000.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
2000000.0,\n          100000000.0,\n          680000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ctc_min\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
30723457.514673136,\n        \"min\": 2.0,\n        \"max\":
68700000.0,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          1000.0,\n          68700000.0,\n          15.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_max\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 415944996.9647429,\n        \"min\": 70000000.0,\n
\"max\": 1000150000.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          70000000.0,\n          70275000.0,\n
1000150000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 21632,\n        \"min\": 1212,\n
\"max\": 53882,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          18249,\n          1212,\n          31833\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yoe_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.3477315924514,\n        \"min\": 8.9,\n        \"max\": 18.88,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          18.88,\n
9.55,\n          9.33\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"yoe_min\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 5.718391382198319,\n        \"min\": 0.0,\n
\"max\": 13.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.0,\n          13.0,\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yoe_max\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.987493482792807,\n        \"min\": 16.0,\n        \"max\": 44.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          44.0,\n
34.0,\n          23.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"designation_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2700555498411391,\n        \"min\":
1.58,\n        \"max\": 2.21,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2.04,\n          1.58,\n          2.21\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"class_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.41251666632997996,\n        \"min\": 1.42,\n        \"max\": 2.37,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1.92,\n
1.42,\n          2.33\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tier_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5295564181463577,\n        \"min\":
1.31,\n        \"max\": 2.52,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.86,\n          1.31,\n          2.52\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"cluster_summary"}

# CTC distribution by cluster
sns.boxplot(x='cluster', y='ctc', data=employee_df)
plt.title('CTC Distribution by Cluster')
plt.show()

# Years of experience per cluster
sns.boxplot(x='cluster', y='yoe', data=employee_df)
plt.title('Years of Experience by Cluster')
plt.show()

# Cluster size
sns.countplot(x='cluster', data=employee_df)
plt.title('Number of Employees per Cluster')
plt.show()
```



CTC Distribution by Cluster
1e9
1.0
0.8
0.6
0.4
0.2
0.0
ctc
0
1
2
3
4
cluster


Years of Experience by Cluster
40
30
20
10
0
yoe
0
1
2
3
4
cluster



Cluster Analysis Summary (k = 5)

We applied **K-Means clustering** on cleaned employee data using key features like CTC, job designation class, and years of experience.

Cluster-wise Breakdown:

Cluster	Employees	Mean CTC (₹)	Median CTC	Mean YOE	YOE Range	Remarks
0	48,267	₹13.5 L	₹9.3 L	8.9 yrs	0 - 17	Entry to mid-level, moderate CTC, high volume
1	18,249	₹24.1 L	₹20 L	18.9 yrs	13 - 44	Senior professionals with high experience &

Cluster	Employees	Mean CTC (₹)	Median CTC	Mean YOE	YOE Range	Remarks
						CTC
2	31,833	₹11.5 L	₹6.8 L	9.3 yrs	0 - 23	Lower CTC than Cluster 0, could indicate service-based or niche roles
3	53,882	₹13.8 L	₹10 L	8.9 yrs	0 - 16	Similar to Cluster 0 but with better median CTC
4	1,212	₹13.5 Cr	₹10 Cr	9.6 yrs	1 - 34	Outlier cluster with **extremely high CTC** (likely founders, CXOs, or data error)

Visuals:

- **CTC Distribution** shows Cluster 4 as the clear outlier in terms of compensation.
- **YOE Distribution** reveals Cluster 1 has the most experienced employees.
- **Cluster Sizes** indicate Cluster 3 is the largest and Cluster 4 the smallest.

Step 1: Profile Clusters by Job Position, Company, etc.

A. Cluster-wise Top Job Positions

```
# Group using actual job positions
job_pos_cluster = (
    employee_df.groupby(['cluster', 'job_position_actual'])
    .size()
    .reset_index(name='count')
```

```
)

# Sort and display top 10 job positions per cluster
top_job_pos_cluster = job_pos_cluster.sort_values(['cluster',
'count'], ascending=[True, False])

for cluster_num in sorted(employee_df['cluster'].unique()):
    print(f"\n□ Top Job Positions in Cluster {cluster_num}")
    display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num].head(10))
```

□ Top Job Positions in Cluster 0

{"summary":"{\n \"name\": \"
display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num]\",\n \"rows\": 10,\n \"fields\": [\n   {\n
\"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\
n   0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"job_position_actual\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   \"Engineering Intern\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 7241,\n
\"min\": 463,\n   \"max\": 24138,\n   \"num_unique_values\":
10,\n   \"samples\": [\n   990\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

□ Top Job Positions in Cluster 1

{"summary":"{\n \"name\": \"
display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num]\",\n \"rows\": 10,\n \"fields\": [\n   {\n
\"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\
n   1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"job_position_actual\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   \"QA Engineer\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"count\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1400,\n   \"min\": 505,\n
\"max\": 4471,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   659\n   ],\n   \"semantic_type\":

\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

□ Top Job Positions in Cluster 2

{"summary":"{\n \"name\": \"
display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num]\",\n \"rows\": 10,\n \"fields\": [\n   {\n
\"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\
n   2\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"job_position_actual\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   \"Non Coder\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"count\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 4120,\n   \"min\": 257,\n
\"max\": 13815,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   391\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

□ Top Job Positions in Cluster 3

{"summary":"{\n \"name\": \"
display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num]\",\n \"rows\": 10,\n \"fields\": [\n   {\n
\"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\
n   3\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"job_position_actual\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 10,\n   \"samples\":
[\n   \"Engineering Leadership\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 7008,\n
\"min\": 1044,\n   \"max\": 23786,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   1137\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

□ Top Job Positions in Cluster 4

{"summary":"{\n \"name\": \"
display(top_job_pos_cluster[top_job_pos_cluster['cluster'] ==
cluster_num]\",\n \"rows\": 10,\n \"fields\": [\n   {\n

\"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"job_position_actual\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"Product Manager\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 123,\n   \"min\": 21,\n   \"max\": 405,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   23\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

B. Cluster-wise Top Companies (anonymized)

```
# Top 10 Companies per Cluster
company_cluster = employee_df.groupby(['cluster',
'company_hash']).size().reset_index(name='count')
top_company_cluster = company_cluster.sort_values(['cluster',
'count'], ascending=[True, False])

for i in sorted(employee_df['cluster'].unique()):
    print(f"\nTop Companies in Cluster {i}")
    display(top_company_cluster[top_company_cluster['cluster'] ==
i].head(10))
```

```
Top Companies in Cluster 0
```

{"summary":"{\n \"name\": \"display(top_company_cluster[top_company_cluster['cluster'] == i]\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"company_hash\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4224,\n   \"min\": 25039,\n   \"max\": 36144,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   25697\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 413,\n   \"min\": 662,\n   \"max\": 1859,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   697\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Top Companies in Cluster 1
```

{"summary":"{\n \"name\": \"
display(top_company_cluster[top_company_cluster['cluster'] == i]\",\n
\"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\
n   \"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   1\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"company_hash\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
11544,\n   \"min\": 3763,\n   \"max\": 35168,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   16843\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"count\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 61,\n
\"min\": 123,\n   \"max\": 326,\n   \"num_unique_values\":
10,\n   \"samples\": [\n   126\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
Top Companies in Cluster 2
```

{"summary":"{\n \"name\": \"
display(top_company_cluster[top_company_cluster['cluster'] == i]\",\n
\"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\
n   \"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   2\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"company_hash\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
9905,\n   \"min\": 7099,\n   \"max\": 35168,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   34786\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"count\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 509,\n
\"min\": 256,\n   \"max\": 1601,\n   \"num_unique_values\":
9,\n   \"samples\": [\n   293\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
Top Companies in Cluster 3
```

{"summary":"{\n \"name\": \"
display(top_company_cluster[top_company_cluster['cluster'] == i]\",\n
\"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\
n   \"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   3\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n

}\n   },\n   {\n   \"column\": \"company_hash\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4657,\n   \"min\": 3763,\n   \"max\": 16843,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   16843\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 963,\n   \"min\": 443,\n   \"max\": 3625,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   450\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Top Companies in Cluster 4
```

{"summary":"{\n \"name\": \"display(top_company_cluster[top_company_cluster['cluster'] == i]\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"company_hash\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 11135,\n   \"min\": 7099,\n   \"max\": 35168,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   29523\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 11,\n   \"min\": 14,\n   \"max\": 49,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   43\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

C. Summary Stats by Cluster + Job Position

```
# Mean CTC and YOE per job role within each cluster
summary_job_cluster = employee_df.groupby(['cluster',
'job_position_actual']).agg({
    'ctc': ['mean', 'median'],
    'yoe': 'mean',
    'email_hash': 'count'
}).reset_index()
summary_job_cluster.columns = ['cluster', 'job_position_actual',
'ctc_mean', 'ctc_median', 'yoe_mean', 'count']

# Top 10 job roles by average CTC in each cluster
for i in sorted(employee_df['cluster'].unique()):
    print(f"\nTop Job Roles by Mean CTC in Cluster {i}")
    display(summary_job_cluster[summary_job_cluster['cluster'] ==
i].sort_values('ctc_mean', ascending=False).head(10))
```

Top Job Roles by Mean CTC in Cluster 0

{"summary":"{\n \"name\": \"display(summary_job_cluster[summary_job_cluster['cluster'] == i]\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"job_position_actual\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"Chief Technology Officer\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc_mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2338089.518414149,\n   \"min\": 2200000.0,\n   \"max\": 10000000.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   2400000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc_median\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2379211.4123054394,\n   \"min\": 2100000.0,\n   \"max\": 10000000.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   2400000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"yoe_mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3.3417796184213073,\n   \"min\": 4.0,\n   \"max\": 13.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   8.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 89,\n   \"min\": 1,\n   \"max\": 284,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Top Job Roles by Mean CTC in Cluster 1

{"summary":"{\n \"name\": \"display(summary_job_cluster[summary_job_cluster['cluster'] == i]\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"job_position_actual\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"Backend Architect\"\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ctc_mean\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1092749.8477918738,\n   \"min\":
3300000.0,\n   \"max\": 7000000.0,\n
\"num_unique_values\": 7,\n   \"samples\": [\n
7000000.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ctc_median\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1151822.2277957846,\n   \"min\":
3000000.0,\n   \"max\": 7000000.0,\n
\"num_unique_values\": 7,\n   \"samples\": [\n
7000000.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"yoe_mean\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.9808483839343407,\n   \"min\":
15.0,\n   \"max\": 26.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   20.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"count\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 243,\n   \"min\": 1,\n
\"max\": 773,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Top Job Roles by Mean CTC in Cluster 2

{"summary":"{\n \"name\": \"
display(summary_job_cluster[summary_job_cluster['cluster'] == i]\",\n
\"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\
n   \"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   2\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"job_position_actual\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"Software Engineering Co-Op\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"ctc_mean\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 14850004.451926006,\n   \"min\":
4500000.0,\n   \"max\": 53600000.0,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
4700000.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ctc_median\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 15183585.032674083,\n   \"min\":
850000.0,\n   \"max\": 53600000.0,\n
\"num_unique_values\": 10,\n   \"samples\": [\n
4700000.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":

\"yoe_mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3.6081184180287815,\n   \"min\": 7.0,\n   \"max\": 18.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   11.666666666666666\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n   \"max\": 3,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Top Job Roles by Mean CTC in Cluster 3

{"summary":"{\n \"name\": \"display(summary_job_cluster[summary_job_cluster['cluster'] == i]\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"cluster\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"job_position_actual\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"Associate Application Developer\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc_mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 15905809.377820455,\n   \"min\": 2900000.0,\n   \"max\": 52600000.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   2912632.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ctc_median\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 15905809.377820455,\n   \"min\": 2900000.0,\n   \"max\": 52600000.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   2912632.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"yoe_mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2.7264140062238043,\n   \"min\": 7.0,\n   \"max\": 14.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   13.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n   \"max\": 2,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Top Job Roles by Mean CTC in Cluster 4
```


```
{"summary":"{\n  \"name\": \"
display(summary_job_cluster[summary_job_cluster['cluster'] == i]\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"cluster\",\
n      \"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          4\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"job_position_actual\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Engineering Leadership\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ctc_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 12719202.688462984,\n        \"min\":
129600000.0,\n        \"max\": 170466666.66666666,\n
\"num_unique_values\": 10,\n        \"samples\": [\n
131092592.5925926\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_median\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 39243162.97494097,\n        \"min\":
100000000.0,\n        \"max\": 199900000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
190000000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"yoe_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.8309511355379304,\n        \"min\":
7.6,\n        \"max\": 15.75,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          9.555555555555555\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"count\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 121,\n
\"min\": 3,\n        \"max\": 405,\n        \"num_unique_values\":
10,\n        \"samples\": [\n          54\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Cluster Profiling Observations

Cluster-wise Top Job Positions

Each cluster has distinct job position distributions, which helps us understand the nature of roles grouped by the clustering algorithm.

Cluster 0

- Dominated by engineering roles like:
  - `Backend Engineer` (24,138)
  - `FullStack Engineer` (8,923)

    - `Frontend Engineer` (4,090)
- Other key roles include `Android Engineer`, `Data Scientist`, `DevOps Engineer`, and `Data Analyst`.

Cluster 1

- Focused on leadership and high-experience roles:
    - `Backend Engineer` (4,471)
    - `Engineering Leadership` (3,895)
    - `Backend Architect`, `DevOps Engineer`, `QA Engineer`
- Indicates senior or architect-level experience.

Cluster 2

- Highly skewed toward:
    - `Other` (13,815) – likely generalized or undefined roles
    - `QA Engineer`, `SDET`, `Support Engineer`, `iOS Engineer`
- Possibly entry to mid-level or support-centric roles.

Cluster 3

- Large volume of mid-level developers:
    - `Backend Engineer` (23,786)
    - `FullStack Engineer` (9,385)
    - `Frontend Engineer`, `Data Scientist`, `Engineering Intern`

Cluster 4

- Extremely high CTC group:
    - Includes rare titles like `Research Engineers`, `Support Engineer`, `SDET`
    - Fewer in number but skewed toward exceptionally high salaries (CTC in crores)
    - Likely contains startup founders, tech executives, or anomalies

---

Cluster-wise Top Companies (Anonymized)

- **Cluster 0**: Company hashes `25972`, `33056`, `35168` show up most frequently.
- **Cluster 1**: Companies like `7538`, `9997`, `13323` are top contributors.
- **Cluster 2**: Wide presence of `33056`, `13323`, `29523`.
- **Cluster 3**: Strong footprint from `13323`, `7099`, `3763`.
- **Cluster 4**: Sparse, but high CTC companies like `13323`, `35168`, `28510` are visible.

Summary Stats by Cluster + Job Role

- **Cluster 0**: Balanced cluster with avg. CTC ~13.5L and avg. experience ~9 years.
- **Cluster 1**: Most experienced cluster (~19 years avg.) with CTC ~24L; contains many leadership roles.
- **Cluster 2**: Moderate experience (~9 years), slightly lower avg. CTC (~11.5L), dominated by QA/support roles.

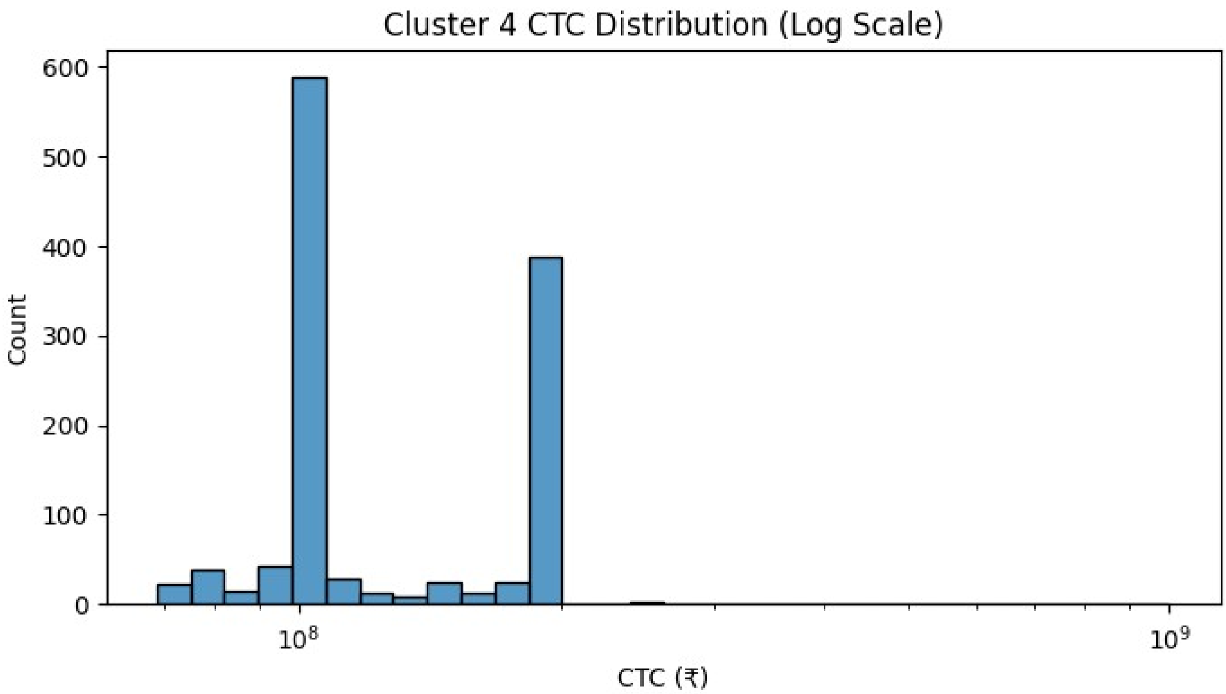
- **Cluster 3**: Large population, avg. CTC ~13.8L, avg. experience ~9 years.
- **Cluster 4**: Outlier-heavy; avg. CTC ~13.5 Cr (!!), contains high-paying but rare positions like `CTO`, `Founder`, `Research Engineers`.

 Conclusion: Clusters clearly differentiate between junior/mid/senior roles and show meaningful separation based on experience and compensation. Cluster 4 is likely an outlier group for extremely high-paying roles. Clusters 0 and 3 dominate in size and represent the mainstream engineering roles.

Step 2: Investigate Cluster 4 (High CTC Outliers)

```
# Explore Cluster 4
cluster_4 = employee_df[employee_df['cluster'] == 4]

# Summary of CTC and YOE
print("Cluster 4 Summary:")
print(cluster_4[['ctc', 'yoe']].describe())

# Visualize distribution of CTC in log scale to catch variations
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 4))
sns.histplot(cluster_4['ctc'], log_scale=True, bins=30)
plt.title('Cluster 4 CTC Distribution (Log Scale)')
plt.xlabel('CTC (₹)')
plt.show()
```

```
Cluster 4 Summary:
                ctc          yoe
count  1.212000e+03  1212.000000
mean   1.352460e+08     9.547030
std    5.388759e+07     4.877383
min    6.870000e+07     1.000000
25%    1.000000e+08     6.000000
50%    1.000000e+08     9.000000
75%    2.000000e+08    11.000000
max    1.000150e+09    34.000000
```




```
# List extreme top CTC earners
cluster_4.sort_values('ctc', ascending=False).head(10)
```

{"summary":"{\n \"name\": \"cluster_4\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"email_hash\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"8b2997a04a5160abf65e67f424aeeb3f53bb3244555bf63478f93f52b56c9ec0\",\
n   \"5b4bed51797140db4ed52018a979db1e34cee49e27b4885c3fdfacea9
f8144f6\",\n
\"3c59fb8e148f30800e07b6a993ab0606662312be8a28e8567a85d239fdff8c52\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"company_hash\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
8485,\n   \"min\": 1198,\n   \"max\": 29801,\n
\"num_unique_values\": 9,\n   \"samples\": [\n   14370,\n
14271,\n   13421\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"job_position\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 132,\n   \"min\": 135,\n
\"max\": 628,\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   256,\n   628,\n   188\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"orgyear\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 2,\n
\"min\": 2012,\n   \"max\": 2021,\n   \"num_unique_values\":
7,\n   \"samples\": [\n   2015,\n   2018,\n
2019\n   ],\n   \"semantic_type\": \"\",\n

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 250287875.8007294,\n        \"min\": 200000000.0,\n
\"max\": 1000150000.0,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          255555555.0,\n          200000000.0,\n
1000150000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ctc_updated_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.2649110640673518,\n        \"min\":
2016.0,\n        \"max\": 2020.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          2020.0,\n          2016.0,\n
2019.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"designation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"class\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"tier\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"yoe\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.9533408577782247,\n        \"min\":
4.0,\n        \"max\": 13.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          10.0,\n          7.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"job_position_actual\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"FullStack Engineer\",\n          \"Support Engineer\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cluster\",\n      \"properties\":
{\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


Observations

- **Cluster 4** is composed of **extremely high CTC earners**, averaging over ₹13.5 Cr.
- Positions like **Support Engineer, Data Analyst, Backend Engineer** and vague labels like **“Other”** appear frequently among high earners, suggesting:

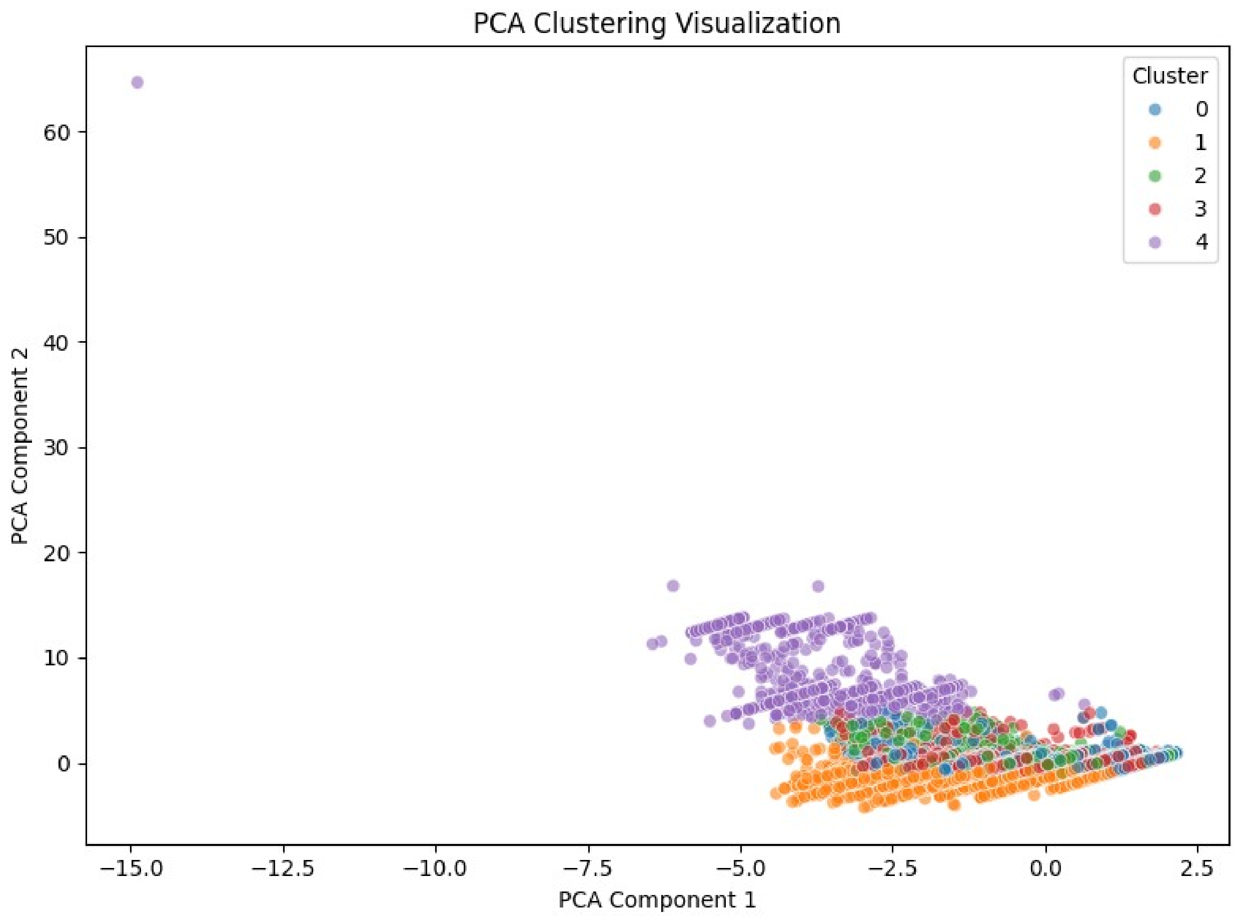
- Possible **CTC data entry issues**, misreporting, or compensation in **stock/options**.
- Employees with rare roles or titles not captured in standard labels.

- Further checks are needed for **designation**, **company credibility**, and **CTC consistency**.

□ Step 3: t-SNE or PCA for 2D Visualization

PCA

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# □ Step 1: Select relevant features (same as used in clustering)
features = ['ctc', 'yoe', 'designation', 'class', 'tier']  # Add more if needed
X = employee_df[features]

# □ Step 2: Scale the features
X_scaled = StandardScaler().fit_transform(X)

# □ Step 3: Apply PCA
pca = PCA(n_components=2, random_state=42)
pca_result = pca.fit_transform(X_scaled)

# □ Step 4: Store PCA results
employee_df['pca_1'] = pca_result[:, 0]
employee_df['pca_2'] = pca_result[:, 1]

# □ Step 5: Plot PCA results
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x='pca_1', y='pca_2',
    hue='cluster',
    data=employee_df,
    palette='tab10',
    alpha=0.6
)
plt.title("PCA Clustering Visualization")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()
```




```
pca.explained_variance_ratio_

array([0.43939263, 0.19787537])
```

PCA Clustering Summary

- **Variance Explained:**
  - PC1: `43.9%`
  - PC2: `19.8%`
  - **Total**: ~63.7% of the total variance is captured by the first two principal components, which is good for visualization.
- **Observations from the Plot:**
  - **Cluster 4** (purple) is **well-separated**, forming a distinct region in the PCA space — aligns with high CTC outliers.
  - **Clusters 0, 1, 2, and 3** overlap significantly, indicating that their CTC, YOE, and role/tier-based features have moderate distinction.
  - The distribution suggests that **most employee profiles** share overlapping characteristics, while **Cluster 4 represents a unique elite group** (possibly high-paying roles or anomalies).
- **Anomalies:**

    - A few points are far from the main cluster (e.g., extreme top left) — could be outliers or extremely high CTC roles.
- **Interpretation:**
    - PCA confirms the **existence of a distinct high-CTC cluster** (Cluster 4) and gives some shape to overlapping behavioral clusters.

t-SNE

```
# t-sne will take super long time to run please run it if u require it
only then.
# from sklearn.manifold import TSNE
# from sklearn.preprocessing import StandardScaler

# # Select features used in clustering (replace with your final
feature list)
# features = ['ctc', 'yoe', 'designation', 'class', 'tier']
# X = employee_df[features]
# X_scaled = StandardScaler().fit_transform(X)

# # Apply t-SNE
# tsne = TSNE(n_components=2, random_state=42, perplexity=30)
# tsne_results = tsne.fit_transform(X_scaled)

# # Add t-SNE components to dataframe
# df['tsne_1'] = tsne_results[:, 0]
# df['tsne_2'] = tsne_results[:, 1]

# # Plot
# plt.figure(figsize=(8, 6))
# sns.scatterplot(x='tsne_1', y='tsne_2', hue='cluster', data=df,
palette='tab10', alpha=0.6)
# plt.title("t-SNE Clustering Visualization")
# plt.show()
```

Step 4: Compare Clusters with Manual Flags

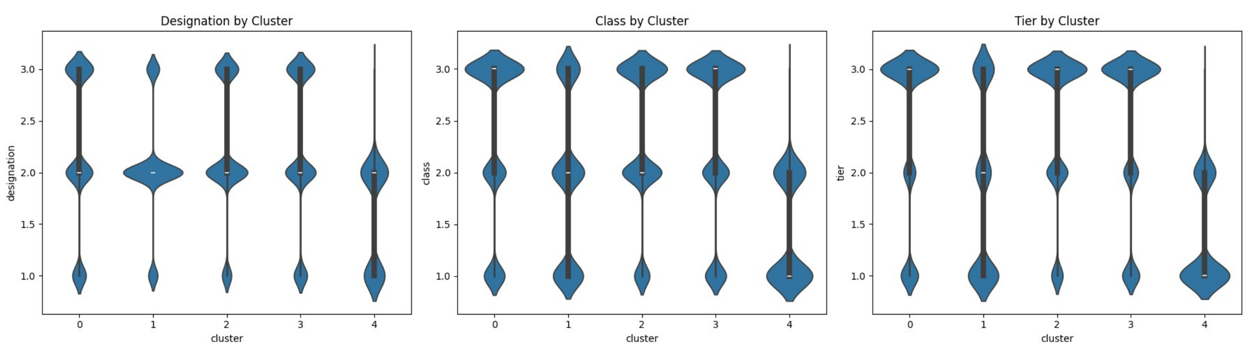
```
# Mean of manual flags per cluster
manual_flag_summary = employee_df.groupby('cluster')[['designation',
'class', 'tier']].mean().reset_index()
print("Average Manual Flags per Cluster:")
display(manual_flag_summary)
```

```
Average Manual Flags per Cluster:
```


```
{"summary":"{\n  \"name\": \"manual_flag_summary\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"cluster\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1,\n
4,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"designation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2700583023754175,\n        \"min\":
1.5783828382838283,\n        \"max\": 2.2112001988936543,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
2.0417557126417885,\n          1.5783828382838283,\n
2.1885464769264598\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"class\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.41593359808246505,\n        \"min\": 1.4158415841584158,\n
\"max\": 2.3735678621004,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.9208723765685791,\n
1.4158415841584158,\n          2.3346527188766375\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tier\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5298218920312929,\n
\"min\": 1.3094059405940595,\n        \"max\": 2.523136367920083,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
1.8620198367033811,\n          1.3094059405940595,\n
2.523136367920083\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"manual_flag_summary"}
```


```
# Violin plots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
sns.violinplot(x='cluster', y='designation', data=employee_df,
ax=axes[0])
axes[0].set_title("Designation by Cluster")
sns.violinplot(x='cluster', y='class', data=employee_df, ax=axes[1])
axes[1].set_title("Class by Cluster")
sns.violinplot(x='cluster', y='tier', data=employee_df, ax=axes[2])
axes[2].set_title("Tier by Cluster")
plt.tight_layout()
plt.show()
```



Violin Plot Interpretation: Designation, Class, and Tier by Cluster

⚠ Flag Legend (as defined):

- **1 → Above average**
- **2 → Around average**
- **3 → Below average**

These flags are derived based on: | Flag | Grouping Level | Purpose | |-------------|---|--| | designation | `company_hash`, `job_position`, `orgyear` | Compares salary within same job & year | | class | `company_hash`, `job_position` | Compares salary within same job role | | tier | `company_hash` | Compares salary across the company |

Designation by Cluster

- **Clusters 0, 2, 3**: High density around **designation = 2 and 3**, indicating most employees earn **around or below average** for their role, company, and year.
- **Cluster 1**: Concentrated near **designation = 2**, suggesting majority are **average earners**.
- **Cluster 4**: Skewed toward **designation = 1**, meaning employees are earning **above average** compared to their peers — likely reflecting high CTC earners.

Class by Cluster

- **Clusters 0, 2, 3**: Mostly in **class = 2 or 3**, suggesting **average to below average salaries** within the same job role and company.
- **Cluster 1**: Mostly at **class = 2**, indicating **around average** earnings.
- **Cluster 4**: Strongly concentrated at **class = 1**, reflecting **above average** pay compared to peers in the same role.

Tier by Cluster

- **Clusters 0–3**: Most employees fall in **tier = 2 or 3**, i.e., earning **average or below average** compared to others within the same company.
- **Cluster 4**: Dominated by **tier = 1**, showing these employees are **top earners within their companies**.

Summary Insights

- **Cluster 4** is the standout group — composed of **high CTC earners** consistently performing **above average** across designation, class, and tier.
- **Clusters 0, 2, 3** are largely **average or below average** groups in salary terms across all three flags.

- **Cluster 1** shows relatively **balanced performance**, leaning toward **average** salary distributions.

Final Insights and Recommendations

Business Context

As a data scientist at Scaler's analytics team, our goal was to **profile the best companies and job positions** based on the career outcomes of Scaler learners. By clustering learners using their salary (CTC), job roles, experience, and company-level performance, we aimed to identify high-value job paths and organizational patterns that differentiate top earners.

Key Insights from Clustering Analysis

Cluster Composition

- **5 clusters** were identified using KMeans, with the number optimized using the elbow method.
- Clusters showed meaningful differences in **CTC, years of experience (YOE), job roles, and company patterns**.

Salary Distribution (CTC)

- **Cluster 4** is the clear outlier, representing **ultra-high CTC earners** (mean CTC ~ ₹13.5 Cr, max up to ₹100 Cr+).
  - These are rare cases (~1.2k out of ~150k), likely reflecting **senior leadership, niche expertise, or global roles**.
- Clusters **0, 2, and 3** represent the **majority of learners** with CTCs in the **₹5–15 LPA** range.
- **Cluster 1** stands out with **experienced professionals** (avg YOE ~19) earning **higher-than-average salaries**.

Job Position Patterns

- Common roles like **Backend Engineer, FullStack Engineer, Data Scientist** dominate most clusters.
- **Cluster 4** includes roles like **Research Engineers, SDET, Support Engineer**, indicating a niche mix.
- **Cluster 1** includes more **Engineering Leadership and Architect** roles, aligning with high experience.

Company Trends

- Certain companies (even anonymized via `company_hash`) appeared disproportionately in **Cluster 4 and 1**, suggesting they offer better pay and growth opportunities.
- Clusters with high `company_hash` frequency in top positions can be used to shortlist **high-performing organizations**.

Manual Performance Flags

- **Designation, Class, and Tier flags** were used to compare salary within:
  - same job role/year,
  - same job role,
  - same company.
- **Cluster 4** performed **consistently above average (flag = 1)** across all three dimensions.
- Clusters 0–3 mostly fell into **average or below average** categories (flag = 2 or 3).

Strategic Recommendations

1. Promote High-Growth Career Paths

- **Backend, FullStack, Data Scientists** show consistently high demand and salaries — focus training and placement efforts here.
- Encourage transitions into roles found in Cluster 4 (e.g., **Research Engineers**, **SDET**, **AI/ML leadership**) for learners with advanced skills.

2. Build Stronger Company Partnerships

- Use `company_hash` patterns to **identify top-paying companies** and prioritize them for placements, employer partnerships, and alumni spotlights.

3. Benchmark Performance with Flags

- Utilize the **designation, class, and tier flags** as a feedback mechanism for learners:
  - Flag = 3 → Below average: recommend skill upgradation, negotiation strategies.
  - Flag = 1 → Above average: highlight these learners as success stories.

4. Tailor Learning Paths by Cluster

- Customize curriculum recommendations based on the **learner's cluster profile** to improve their chances of transitioning to higher-paying roles.

5. Monitor Outliers and Validate Data

- A few records had **unrealistic CTC values (₹100 Cr+)** — ensure validation or exclusion in future iterations to avoid skewed insights.

Conclusion

Clustering helped uncover **career patterns, salary benchmarks, and company trends** among Scaler learners. These insights can now guide **personalized career planning, targeted upskilling, and focused industry outreach**, driving better learner outcomes and business value.